INRIA

CENTRE DE ROCQUENCOURT

Rapports Techniques

N° 10

OUTILS DE MESURES DE PROGRAMMES PASCAL

MANUEL D'UTILISATION

Anne SCHROEDER

Avril 1982

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. 954 90 20

OUTILS de MESURES de PROGRAMMES PASCAL
- Manuel d'utilisation -

Anne SCHROEDER

Résumé :

Voici le mode d'emploi d'un certain nombre d'outils destinés à mesurer des programmes Pascal. Les mesures envisagées sont soit statiques, c'est-à-dire relatives au code même du programme, soit dynamiques, c'est-à-dire prélevées à l'exécution.

Tous les outils présentés dans cette note ont été développés sous Mentor, et donc écrits en Mentol. Toutefois, ils peuvent être utilisés par n'importe quel utilisateur Pascal travaillant sous Multics à l'INRIA, même s'il n'est pas adepte de Mentor. Ils sont également exportables sur tout site où Mentor tourne.

La naissance récente de Meta-Mentor (version multi-langage de Mentor) permet d'espérer l'extension des actuels outils Pascal à d'autres langages.

Abstract :

This paper is a user's manual for some measurement tools on Pascal programs. The measurements involved can be either static, i.e. taken at source level, or dynamic, i.e. gathered at run time.

All the tools presented in this note have been developped using the Mentor programming environment. However, they can be used by any Pascal user on Multics-INRIA even if he does not work under Mentor. They also are portable on any site supporting Mentor.

The new arrival of a multi-language version of Mentor may allow to extend the existing tools dedicated to Pascal to other languages.

OUTILS de MESURES de PROGRAMMES PASCAL

- Manuel d'utilisation -

Anne SCHROEDER

- Mars 1982 -

1 - Préliminaire

Ceci est le mode d'emploi d'un certain nombre d'outils destinés à mesurer des programmes Pascal. Les mesures envisagées sont soit statiques, c'est-à-dire relatives au code même du programme, soit dynamiques, c'est-à-dire prélevées à l'exécution.

Tous les outils présentés dans cette note ont été développés sous Mentor[2,8], et donc écrits en Mentol.

Ils peuvent être utilisés par trois types d'usagers de Pascal :

a) ceux qui ne veulent pas entendre parler de Mentor, pour lesquels on a écrit des commandes qui cachent soigneusement le système;

b) ceux dont les programmes sont déjà développés sous Mentor (ou qui acceptent de les y mettre...), mais qui veulent se donner le moins de mal possible; pour ceux-ci ont été écrites des procédures Mentol «encapsulantes» qui font tout le travail;

c) les utilisateurs habituels de Mentor qui voudront tout savoir, auxquels on fournira l'ensemble des procédures Mentol disponibles et qui se débrouilleront pour les utiliser à bon escient.

Ce manuel comprend une description des différentes mesures que l'on peut effectuer sur des programmes Pascal suivie de modes d'emploi personnalisés adaptés aux usagers a), b) ou c).

Tous les fichiers utiles pour se servir des outils proposés se trouvent dans

```
>udd>Mmsi>Schroeder>lib;
```

il suffit pour les usagers a) de se mettre une search-rule sur ce directory :

```
asr  >udd>Mmsi>Schroeder>lib
```

tandis que les usagers b) et c) ajouteront le search-path suivant :

```
asp  mentor >udd>Mmsi>Schroeder>lib
```

et chargeront la bibliothèque de procédures Mentol :

```
ASENV.mentol
```

soit par .xin dans leur session Mentor, soit en l'intégrant dans leur prélude personnel.

2 - Mesures statiques de complexité des programmes

2.1 - Description des mesures

Un certain nombre de mesures de complexité d'un programme est proposé dans la littérature; il s'agit de mesures statiques en ce sens qu'elles ne dépendent que du texte du programme et sont donc indépendantes de son comportement à l'exécution. On peut les classer en deux groupes :

- les mesures textuelles qui portent sur la syntaxe du programme : nombres d'opérateurs et d'opérandes distincts ou non, distribution des opérateurs, profondeur de l'arbre syntaxique, références aux variables, etc... [3,4,5,11]
- les mesures portant sur la structure logique du programme, qui sont des caractéristiques du graphe de contrôle : nombre de nœuds et de chemins, niveau d'imbrication des boucles, nombre cyclomatique (ou nombre de branchements) [7,9,10].

Les outils proposés calculent ces mesures pour chaque bloc composant le programme soumis à l'analyse. Les mesures fournies sont plus précisément les suivantes :

- la profondeur de l'arbre syntaxique;
- les nombres d'opérateurs et d'opérandes, distincts et non distincts. La définition des opérateurs et des opérandes pour un langage donné est sujette à quelques ambiguïtés, aussi précisons-nous le choix qui a été le nôtre : les opérandes sont tous les nœuds terminaux de l'arbre syntaxique à l'exception des noms des procédures appelées, tandis que les opérateurs sont les nœuds non terminaux auxquels on ajoute les noms des procédures appelées;
- les distributions des opérateurs et des opérandes (la liste des opérateurs distincts étant la seule donnée);
- trois variantes du nombre cyclomatique : 1) nb.cyc.min étant calculé en ne comptant qu'un branchement pour les tests IF et CASE 2) nb.cyc.mid comptant tous les branchements dûs aux CASE 3) nb.cyc.max comptant également les branchements dûs aux IF lorsque la condition est composée;
- les niveaux d'imbrication moyen et maximum des boucles;
- les nombres de nœuds et de chemins du graphe de contrôle du programme dans lequel sont réunis en un même nœud tous les ensembles purement sèquentiels d'instructions; le nombre de nœuds de ce graphe est parfois appelé nombre cocyclomatique du graphe; ces deux mesures permettent de calculer l'atteignabilité moyenne du programme : nombre de chemins ÷ nombre de nœuds;
- le taux de références aux variables dans chaque bloc : quotient du nombre total de références aux variables dans le texte par le nombre de variables déclarées localement ou globalement; on donne également les listes des variables locales et globales avec les nombres de références à chacune.

Notons au passage l'existence dans l'environnement standard Mentor[8] de quelques procédures fournissant une information statique sur les appels de fonctions et de procédures (graphes d'appels vus sous plusieurs angles différents, détection des récursions croisées).

2.2 - Mise en œuvre

Utilisateurs a)

Si TOTO est le nom du programme Pascal à mesurer (suivi ou non de son suffixe «.pascal», mais, impérativement en majuscules), on utilisera la commande suivante :

```
messtat TOTO
```

qui donnera, pour chaque bloc composant le programme mesuré, tous les résultats décrits au paragraphe précédent.

Utilisateurs b)

Les deux procédures Mentol suivantes donnent tous les résultats mentionnés ci-dessus :

.SYNCONTR1	mesures prises sur l'arbre syntaxique et sur le graphe de contrôle,

.REFVSTAT	statistiques portant sur les références aux variables.

Utilisateurs c)

Liste des procédures Mentol disponibles ayant trait aux mesures statiques des programmes (outre les deux procédures indiquées aux utilisateurs b)) :

.GRCONTR	mesures relatives au graphe de contrôle (nombre cyclomatique et atteignabilité)
.ARSYNT	mesures relatives à l'arbre syntaxique (profondeur, opérateurs et opérandes)
.IMBRIC	Distribution des niveaux d'imbrication des boucles (dans le texte du programme, le niveau d'imbrication de chaque boucle est indiqué en commentaire préfixe de la dite boucle)
.SYNCONTR2<@nn>	mêmes mesures que .SYNCONTR1 avec une sortie abrégée et une numérotation des procédures mesurées, @nn étant la valeur initiale de la numérotation.

Cette dernière procédure est utile lorsqu'on veut constituer un fichier de mesures portant sur un grand nombre de blocs Pascal.

3 - Mesures dynamiques ou traces à l'exécution des programmes

3.1 - Description des outils de trace

Les outils fournis permettent de tracer l'exécution de certaines instructions. Sur l'intérêt de telles traces, consulter [6] et [1]. Ces traces peuvent être soit immédiates, soit cumulées sur plusieurs exécutions. On peut également ne tracer un certain type d'instruction que dans une partie d'un programme (cette partie doit en réalité être un sous-arbre Mentor).

D'une façon générale, les traces se présentent de la manière suivante : le nombre de fois où une instruction tracée a été exécutée s'inscrit dans le texte du programme en commentaire préfixe de la dite instruction.

A l'instant présent, nous savons tracer les instructions suivantes :

- les itérations de boucles FOR, WHILE et REPEAT; les compteurs sont alors placés en tête de l'instruction qui constitue le corps de la boucle;
- les têtes de blocs, ou plus précisément les entrées dans le corps d'une procédure ou d'une fonction;
- les appels aux procédures et fonctions internes et externes; les compteurs sont placés devant chaque appel et, en fin de texte, on trouve un récapitulatif qui donne les fréquences d'appel de chaque procédure;
- les tests IF et CASE; pour les premiers, on a un compteur devant le IF et un autre en tête de l'instruction constituant le THEN; pour les seconds, un compteur précède chacune des instructions constituant les alternatives du CASE.

Il va sans dire que l'on pourrait tracer finalement n'importe quel type d'instruction avec des outils très proches de ceux existants; si vous avez un désir particulier dans ce domaine, adressez-vous à l'auteur.

On peut tracer plusieurs types d'instructions au cours d'une exécution, c'est-à-dire à l'aide des procédures que nous avons appelées procédures de traces immédiates; par contre, on ne saura représenter de traces cumulées que pour un type d'instruction à la fois, ce qui n'oblige pourtant pas à exécuter le programme autant de fois qu'il y a de types d'instructions à tracer (détails dans 3.2.c)).

Notons à nouveau l'existence dans l'environnement standard Mentor[8] de quelques procédures de trace (traces d'une instruction donnée, des entrées dans les procédures et les fonctions ainsi que des appels).

3.2 - Mise en œuvre

Utilisateurs a)

On utilisera les commandes suivantes :

traceimm TOTO arg*1* arg*2* ...arg*n* pour une trace immédiate,

ou

tracecum TOTO arg*1* pour une trace cumulée,

où TOTO est le nom du programme Pascal à tracer (avec ou sans son suffixe «.pascal», mais en majuscules... si cette contrainte vous indipose, adressez-vous aux responsables Mentor), et où les arg*i* indiquent quels types d'instructions sont à exécuter; au jour d'aujourd'hui ils peuvent être égaux à un des mots-clés suivants : boucle, bloc, appel ou test.

De la commande traceimm, tous les fichiers sortent indemnes, mis à part l'exécutable du programme tracé qui est remplacé par l'exécutable de la version instrumentée; en conséquence, si on veut exécuter à nouveau le programme initial, il faudra le recompiler -son source est intact-. Attention : si ce qui précède n'est pas vraiment faux, ce n'est pas tout-à-fait vrai non plus; en réalité, le source du programme a été *pretty-printed* (ndtr : *joliment imprimé*, comme ce manuel); en principe, ça le rend plus joli et ne peut donc pas nuire, pourtant si vous tenez à votre texte source et ne pouvez admettre de le voir modifié d'un ι(!), faites en une copie quelque part.

A la sortie de la commande tracecum, si le programme tracé avait pour nom TOTO, on se retrouve à la tête du fichier suivant à garder soigneusement, si on veut cumuler les traces sur plusieurs exécutions :

TOTO1.polish : représentation de la trace cumulée de TOTO.

Utilisateurs b)

Les procédures Mentol de relevés de traces s'appellent ainsi :

BIG*TRUC* pour une trace immédiate,

ou

BIG*TRUC*C pour une trace cumulée,

le mot-clé *TRUC* pouvant prendre une des valeurs suivantes : BOUCLE, BLOC, APPEL ou TEST, selon le type d'instruction à tracer.

Des procédures BIG*TRUC* tous les fichiers sortent indemnes, mis à part l'exécutable du programme tracé qui est remplacé par l'exécutable de la version instrumentée; en conséquence, si on veut exécuter à nouveau le programme initial, il faudra le recompiler -son source est intact (consultez pourtant la mise en garde destinée aux usagers a) ci-dessus)-; en prime, on vous offre, si le programme tracé avait pour nom TOTO, un fichier TOTOOUT.mentol qui contient les relevés de trace et qui est bon à détruire en général.

A la sortie des procédures BIG*TRUC*C, si le programme tracé avait pour nom TOTO, on se retrouve à la tête des fichiers suivants (à garder soigneusement, si on veut vraiment cumuler les traces sur plusieurs exécutions) :

TOTO0.polish : programme initial;
TOTO.polish : version instrumentée;
TOTO.pascal : décompilation de la version instrumentée;
toto : module objet de la version instrumentée;
TOTO1.polish : représentation de la trace cumulée de TOTO;
et, bien sûr, TOTOOUT.mentol : fichier des relevés de compteurs dont on peut disposer à son aise...

Utilisateurs c)

Pour les usagers avertis que sont les utilisateurs *c)*, on se contentera de donner la liste des procédures Mentol disponibles.

Pour tracer l'entité *TRUC* :
Bibliothèque contenant les procédures spécifiques à la trace de *TRUC* : BIB*TRUC*.mentol

Valeurs que peut prendre le mot-clé *TRUC*	BOUCLE BLOC APPEL TEST

Procédures d'instrumentation des programmes

.*TRUC*S	Mise en place des compteurs, qui sont les éléments d'un tableau *TRUC* - une exception : pour tracer les tests, il y a deux tableaux de compteurs TEST et CAS-.

Diverses initialisations sont effectuées par appel aux deux procédures suivantes :

.INITCT	Mise en place des ordres concernant le fichier de relevés de traces : création, ouverture, fermeture. Cette procédure est indépendante de l'entité tracée et n'est appelée qu'à la première demande d'instrumentation d'un programme donné. En conséquence, elle ne se trouve pas dans BIB*TRUC*.mentol, mais dans la bibliothèque générale ASENV.mentol.
.*TRUC*INIT	Appel aux procédures Pascal d'initialisation des compteurs (INI*TRUC*) et de sortie (OUT*TRUC*) sur TOTOOUT.mentol. Les procédures Pascal en question se trouvent dans le fichier SOUSP.pascal.

Procédures de présentation de la trace

.*TRUC*TRACE	Trace immédiate
.*TRUC*TRCUM	Trace cumulée. Cette procédure suppose l'existence d'un fichier TOTO1.polish contenant l'état actuel de la trace cumulée; avant le premier appel à .*TRUC*TRCUM, il faut donc créer un fichier TOTO1.polish initial à l'aide de la procédure suivante;
.*TRUC*TRC0	Création d'un fichier initial de trace cumulée à partir du programme de départ TOTO0.polish .

Procédures «encapsulantes» qui instrumentent un programme, l'exécutent et en donnent la trace

.BIG*TRUC*	Trace immédiate
.BIG*TRUC*C	Trace cumulée.

Nous donnons maintenant un bref organigramme de .BIG*TRUC* et de .BIG*TRUC*C, pour indiquer la façon logique d'utiliser les procédures décrites ci-dessus.

Procédure .BIG*TRUC* :

Un programme TOTO étant chargé

Stocker TOTO dans TOTO0.polish

.*TRUC*S

Compilation

Exécution

Chargement de TOTO0.polish
Chargement de TOTOOUT.mentol

.*TRUC*TRACE

Procédure .BIG*TRUC*C :

S'il s'agit effectivement de la n-ième exécution
d'une trace cumulée, les fichiers TOTO0.polish ,
TOTO.polish , et toto sont
supposés contenir ce qu'il faut....
Sinon, on commence par stocker le programme initial
dans TOTO0.polish , puis on instrumente
et on compile comme ci-dessus

Exécution

Chargement de TOTOOUT.mentol

Chargement de TOTO1.polish
- s'il existe, ok,
- sinon, charger TOTO0.polish et exécuter .*TRUC*TRC0

.*TRUC*TRCUM

Si on veut effectuer simultanément la trace de plusieurs types d'instructions *TRUC1,TRUC2*,... (ainsi que le fait la commande traceimm - cf.4.), on procèdera de la façcon suivante :

Un programme TOTO étant chargé

Stocker TOTO dans TOTO0.polish

.*TRUC1*S
.*TRUC2*S
...

Compilation

Exécution

Chargement de TOTO0.polish
Chargement de TOTOOUT.mentol

.*TRUC1*TRACE
.*TRUC2*TRACE
...

4 - Exemples

```
program TOTO(OUTPUT);
     var  I,J,IMAX:INTEGER;
          FICH:TEXT;
     begin
     FCONNECT(FICH,'vfile  >udd>Mmsi>Schroeder>
                           pascd>pas>donn');
     RESET(FICH);
     READ(FICH,IMAX);
     for I:=1 to IMAX do
          begin
          WRITELN('ouh-ouh!');
          if(I<2)or(I>3) then
               begin
               WRITELN('Youp-la...!...')
               end
         end;
     I:=1;
     while I<=IMAX do
          begin
          WRITE('ooooooouh...');
          J:=0;
          repeat
               WRITELN('whaouh!');
               J:=J+1
          until J=2;
          I:=I+1
          end;
     FCLOSE(FICH);
     end.
```

Texte du programme test à mesurer

```
messtat TOTO
- MENTOR  VERSION 4.1  JAN-82  ON MULTICS (IREP)
USER NAME:SCHROEDER.prelude FILE LOADING
?FILE NAME:?++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++
program TOTO(OUTPUT)
Profondeur
7
nb.operandes
33
nb.operandes distincts
14
nb.operateurs
45
nb.operateurs distincts
21
Distribution des operateurs
(TEXT,LSTAT,CALL,FCONNECT,LEXP,RESET,READ,FOR,
 UPSTEP,WRITELN,IF,OR,LSS,GTR,ASS,
 WHILE,LEQ,WRITE,REPEAT,PLUS,EQL,FCLOSE)
(0,5,8,1,8,1,1,1,1,3,1,1,1,1,4,1,1,1,1,2,1,1)
Distribution des operandes
(0,4,1,3,7,4,1,2,1,1,2,1,4,1,1)
nb.cyclomatique min.
```

Exécution de la commande MESSTAT (sorties de .SYNCONTR1 et .REFVSTAT.)

Comme on le voit, on ne peut pas faire taire Mentor tout-à-fait et il envoie quelques reliques. Tant pis...

```
4
nb.cyclomatique mid.
4
nb.cyclomatique max.
5
atteignabilite: nb.noeuds
15
----------------nb.chemins
20
??++++++++++++++++++++++++++++++++++
program TOTO(OUTPUT)
++++++++++++++++++++++++++++++++++
Taux de reference aux variables :
18 divise par 4
++++++++++++++++++++++++++++++++++
nb. de references aux variables locales :
(I,J,IMAX,FICH)
(7,4,3,4)
++++++++++++++++++++++++++++++++++

Niveau d'imbrication maximum des boucles :
2
Niveau d'imbrication moyen des boucles :
4 divise par 3
?EXIT LEV.1-PASCAL
```

```
traceimm TOTO appel boucle
- MENTOR  VERSION 4.1  JAN-82  ON MULTICS (IREP)
USER NAME:SCHROEDER.prelude FILE LOADING
?FILE NAME:
?[:?[:?[:?[:?[:?[:?
TOTOO.polish  FILE OVERWRITTEN
TOTO.polish  FILE OVERWRITTEN
TOTO.pascal  FILE OVERWRITTEN
PASCAL 6.02
```

Exécution de la commande TRACEIMM, appelée avec les deux arguments appel et boucle

```
ouh-ouh!
Youp-la...!...
ouh-ouh!
ouh-ouh!
ouh-ouh!
Youp-la...!...
 e
  t
   c...
ooooooouh...whaouh!
whaouh!
TOTO.pascal  FILE OVERWRITTEN
```

Sorties (intéressantes) du programme TOTO

```
program TOTO(OUTPUT);
    var  I,J,IMAX:INTEGER;
       FICH:TEXT;
    begin
    (*1*)
```

Trace de TOTO

Certains compteurs font double-emploi; ceci vient de ce que les traces de deux types

```
    FCONNECT(FICH,'vfile  Oudd>Mmsi>Schroeder>
                    pascd>pas>donn);
    (*1*)
    RESET(FICH);
    (*1*)
    READ(FICH,IMAX);
    for I:=1 to IMAX do (*8*)
        begin
        (*8*)
        WRITELN('ouh-ouh!');
        if(I<2)or(I>3) then
            begin
            (*6*)
            WRITELN('Youp-la...!...')
            end
        end;
    I:=1;
    while I<=IMAX do (*8*)
        begin
        (*8*)
        WRITE('ooooooouh...');
        J:=0;
        repeat
            (*16*)
            (*16*)
            WRITELN('whaouh!');
            J:=J+1
        until J=2;
        I:=I+1
        end;
    (*1*)
    FCLOSE(FICH);;
    end (*
    begin
    FCLOSE:=1;
    WRITE:=8;
    WRITELN:=30;
    READ:=1;
    RESET:=1;
    FCONNECT:=1
    end*).
?EXIT LEV.1-PASCAL
```

d'instruction différents ont été demandés. Par exemple, dans la boucle REPEAT, le compteur donnant la valeur 16 intervient d'une part en préfixe du corps de la boucle, et d'autre part, en préfixe de l'appel à procédure qu'est le WRITELN.

A la fin de la trace figure un récapitulatif des appels effectifs aux différentes procédures.

Voici un nouveau programme test...

```
program TOTOCASE(OUTPUT)
     var  I,J:INTEGER;
     begin
     for I:=1 to 10 do
          begin
          if(I<5)and(I>2) then
               begin
               WRITELN('Ouh-Ouh')
               end
          else begin
               WRITELN('Poum!')
```

```
            end;
        case I of
            2,4,6,8:
                begin
                WRITELN(' i est pair')
                end;
            1,3,7,9:
                begin
                WRITELN(' i est impair')
                end;
            5:  begin
                WRITELN(' Youpee ! I vaut 5...')
                end;
            10: begin
                WRITELN(' Bravo... I vaut 10 !')
                end
        end
    end.
```

```
tracecum TOTOCASE test
- MENTOR  VERSION 4.1  JAN-82  ON MULTICS (IREP)
USER NAME:SCHROEDER.prelude FILE LOADING
?[:?[:??[:?TOTOCASEO.polish  FILE OVERWRITTEN
lline
TOTOCASE.polish  FILE OVERWRITTEN
TOTOCASE.pascal  FILE OVERWRITTEN
PASCAL 6.02
Poum!
i est impair
Poum!
i est pair
Ouh-Ouh
i est impair
Ouh-Ouh
i est pair
Poum!
Youpee ! I vaut 5...
Poum!
i est pair
Poum!
i est impair
Poum!
i est pair
Poum!
i est impair
Poum!
Bravo... I vaut 10 !
```

Premier appel à la commande TRACECUM avec l'argument test.

```
program TOTOCASE(OUTPUT);
    var  I,J:INTEGER;
(*1*)
    begin
    for I:=1 to 10 do
        begin
```

On remarque ici le commentaire indiquant le nombre d'exécutions du

```
        (*10*)
        if(I<5)and(I>2) then (*2*)
            begin
            WRITELN('Ouh-Ouh')
            end
        else begin
            WRITELN('Poum!')
            end;
        case I of
            2,4,6,8: (*4*)
                begin
                WRITELN(' i est pair')
                end;
            1,3,7,9: (*4*)
                begin
                WRITELN(' i est impair')
                end;
            5: (*1*)
                begin
                WRITELN(' Youpee ! I vaut 5...')
                end;
            10: (*1*)
                begin
                WRITELN(' Bravo....I vaut 10 !')
                end
        end
        end
    end.
TOTOCASE1.polish  FILE CREATED
?EXIT LEV 1-PASCAL
```

programme prises en compte dans le cumul de la trace.

```
tracecum TOTOCASE test
- MENTOR  VERSION 4.1  JAN-82  ON MULTICS (IREP)
USER NAME:SCHROEDER.prelude FILE LOADING
?[:?[:??[:?Poum!
i est impair
Poum!
 e
  t
   c...
Bravo...I vaut 10 !
```

Deuxième appel à la commande de trace cumulée

```
program TOTOCASE(OUTPUT);
    var  I,J:INTEGER;
(*2*)
    begin
    for I:=1 to 10 do
        begin
        (*20*)
        if(I<5)and(I>2) then (*4*)
            begin
            WRITELN('Ouh-Ouh')
            end
        else begin
```

```
            WRITELN('Poum!')
            end;
        case I of
            2,4,6,8: (*8*)
                begin
                WRITELN(' i est pair')
                end;
            1,3,7,9: (*8*)
                begin
                WRITELN(' i est impair')
                end;
            5: (*2*)
                begin
                WRITELN(' Youpee ! I vaut 5...')
                end;
            10: (*2*)
                begin
                WRITELN(' Bravo... I vaut 10 !')
                end
        end
        end
    end.
TOTOCASE1.polish  FILE OVERWRITTEN
?EXIT LEV.1-PASCAL
```

5 - Bibliographie

[1] Jacques Cohen and Neal Carpenter
Language for Inquiring about the Run-time Behaviour of Programs
Software-Practice and Experience
Vol.7, 445-460 (1977)

[2] V. Donzeau-Gouge, G. Huet, G. Kahn, B. Lang and J.-J. Lévy
A Structure Oriented Program Editor : a First Step toward Assisted Programming
International Computing Symposium, North-Holland pub.
(1975)

[3] Ann Fitzsimmons and Tom Love
Review and Evaluation of Software Science
Computing Surveys,
Vol.10, No.1, 3-18 (1978)

[4] M.H. Halstead
Elements of Software Science
Elsevier North-Holland, Inc., N.Y.
(1977)

[5] Wilfred J. Hansen
Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)
Sigplan Notices, 13(3), 29-33
(March 1978)

[6] Donald E. Knuth
An Empirical Study of Fortran Programs
Software-Practice and Experience,
Vol.1, 105-133
(1971)

[7] Thomas J. McCabe
A Complexity Measure
IEEE Transactions on Software Engineering,
Vol.SE-2, No.4, 308-320
(December 1976)

[8] Bertrand Mélèse
Mentor, l'environnement Pascal
INRIA - Rapport Technique no.5
(Octobre 1981)

[9] Glenford J. Myers
An Extension to the Cyclomatic Measure of Program Complexity
Sigplan Notices 12(10), 61-64
(October 1977)

[10] N.F. Schneidewind and Heinz-Michael Hoffmann
An Experiment in Software Error Data Collection and Analysis
IEEE Transactions on Software Engineering,
Vol SE-5, No.3, 276-286
(May 1979)

[11] Jean M. Zolnowski and Dick B. Simmons
Measuring Program Complexity
Digest of Papers of Fall COMPCON77,
IEEE Cat. No.77CH1258-3C, 336-3
(1977)

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique